VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ PODOBNÝCH 3D MODELŮ

SEARCHING FOR SIMILAR 3D MODELS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MIROSLAV KARÁSEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL ŠPANĚL, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Karásek Miroslav**
Obor: Informační technologie
Téma: **Vyhledávání podobných 3D modelů**
Searching for Similar 3D Models
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte dostupné materiály na téma popis polygonálních modelů a tvaru 3D objektů (tzv. shape descriptors). Seznamte se s metodami jejich sesouhlasení a porovnání.
2. Vyberte vhodné metody a navrhněte nástroj pro vyhledávání 3D modelů v databázi, kdy dotazem je ukázkový 3D model.
3. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace metod.
4. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
5. Vytvořte stručný plakát nebo video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních dvou bodů zadaní.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Španěl Michal, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se zabývá návrhem a implementací vyhledávání 3D modelů v databázi na základě podobnosti. Samotná databáze je realizována za použití rychlých datových struktur v jazyce Python. Využívá knihovny třetích stran na extrakci Spherical harmonics a 3D Zernikeho deskriptorů. Pro přístup k databázi je navrženo REST aplikační rozhraní, jehož využití je demonstrováno na ukázkovém webovém uživatelském rozhraní. Dále se práce zabývá kvalitou vyhledávání, prezentuje výsledky implementovaného vyhledávacího enginu a navrhuje možná vylepšení.

Abstract

The work deals with the design and implementation of similarity based search in database of 3D models. The database is realised using of quick Python data structures. For Spherical harmonics and 3D Zernike deskriptors extraction are used third-party libraries. REST application interface is designed to access the search engine. A web based application was designed as an example of the REST application interface usage. Furthermore, the work deals with the search quality, presents results of the implemented database and suggests possible improvements.

Klíčová slova

Vyhledávání, podobnost, 3D tvar, 3D model, Spherical harmonics, Zernikeho deskriptor, databáze, Python.

Keywords

Searching, similarity, 3D shape, 3D model, Spherical harmonics, Zernikeho descriptor, database, Python.

Citace

KARÁSEK, Miroslav. *Vyhledávání podobných 3D modelů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Španěl Michal.

Vyhledávání podobných 3D modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Karásek
16. května 2017

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Michalovi Španělovi, Ph.D. za odborné vedení a cenné rady při psaní této práce.

Obsah

Kapitola 1

Úvod

Náš svět je trojrozměrný prostor a počítačové systémy mohou díky neustále rostoucímu výpočetnímu výkonu tuto realitu čím dál věrněji napodobovat. Technologie tzv. virtuální reality nám umožňuje být v podstatě součástí scény generované počítačem. Proto je nutné zabývat se technologiemi, které umožní počítači provádět pro člověka přirozené rozhodování, jako je například vyhledávání mezi 3D modely. Podobně jako nyní počítačové systémy lidem usnadňují vyhledávání textu, což bychom mohli nazvat 1D prostorem. Vyspělé technologie ovšem existují i pro vyhledávání v 2D prostoru, například v obrázcích. S přibývajícími rozměry je ale výrazně složitější napodobit lidské vnímáni pomocí algoritmů.

Cílem práce je navrhnout a v Pythonu implementovat engine pro vyhledávání 3D modelů v jednoduché databázi. Vyhledávání bude uskutečněno na základě podobnosti dvou modelů. Nepůjde tedy o běžné 1D vyhledávání, tj. podle popisu či tagů. Dotazem zde bude jeden konkrétní model databáze. Odpovědí, kterou zpracuje implementovaný engine, bude seznam obsahující modely podobné dotazovanému.

Bude využita architektura klient-server, kde v roli serveru bude navrhovaná databáze a vyhledávácí engine. Klient může být libovolný program, který využije funkcionalit této databáze. Pro komunikaci bude navrženo REST aplikační rozhraní (API). Práce se zaměřuje nejvíce na server a jeho rozhraní. Klient je vytvořen pouze pro jednoduchou a názornou demonstraci výsledné práce pomocí grafického uživatelského rozhraní.

K porovnání 3D modelů a potažmo jejich vyhledávání existují již navržené metody. Navrhovaný engine se zaměřuje na metodu *Spherical harmonics* a metodu tzv. *3D Zernikeho deskriptorů*. Pro obě jsou využity existující knihovny určené pro jazyk C++.

Práce je členěna do několika kapitol. První dvě shrnují teorii potřebnou k porozumění obsahu práce. Další část se zabývá dekompozicí a návrhem výsledné aplikace. Návrh je následně realizován. Implementační detaily realizovaného návrhu jsou detailně popsány v další kapitole. V závěru práce je uvedeno zhodnocení kvality výsledného vyhledávacího enginu spolu s návrhem možných vylepšení.

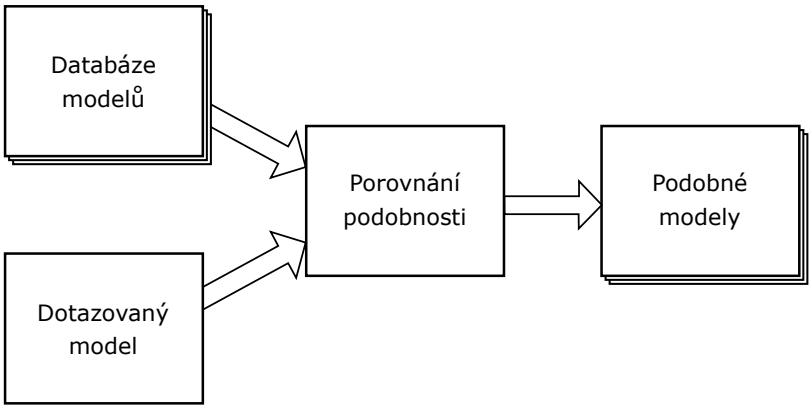
Kapitola 2

Metody vyhledání 3D tvaru založené na obsahu

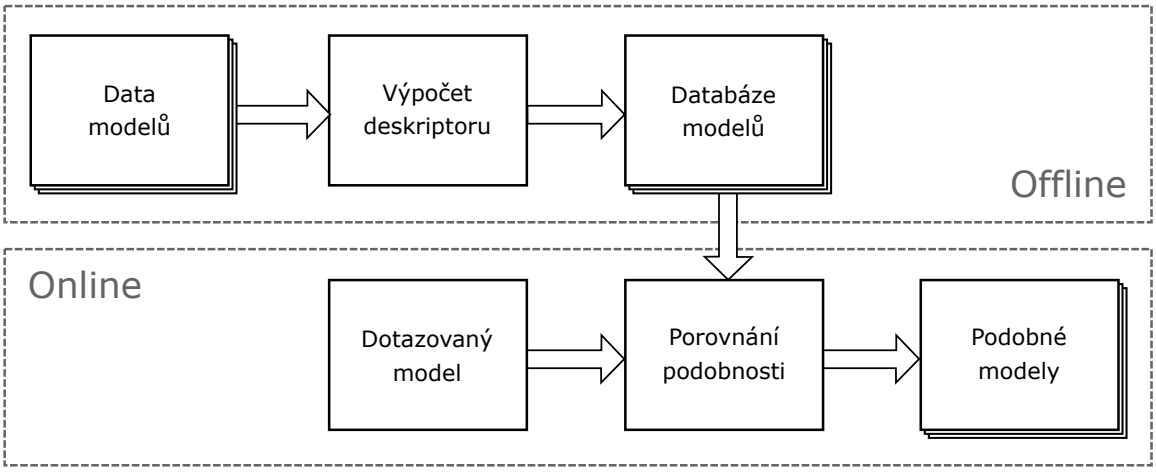
Před tím, než se budeme zabývat návrhem vlastního enginu pro vyhledávání podobných 3D modelů, bude třeba se seznámit s teorii z této oblasti. Kapitola nejprve nastiňuje obecný princip vyhledávání (viz podkapitola 2.1), který se používá nejen při porovnávání a vyhledávání 3D modelů. Ve zbylých podkapitolách jsou podrobně popsány další důležité pojmy, které jsou potřebné pro správné pochopení problematiky.

2.1 Princip vyhledávání

Uvažujme, že máme množinu 3D modelů M a jako dotaz pro vyhledání podobných modelů použijeme jeden prvek q této množiny. Abychom získali podobné modely, potřebujeme znát podobnost prvku q se všemi prvky množiny M. Podobnost je podrobněji popsána v podkapitole 2.2. V případě, že tuto podobnost známe, stačí prvky seřadit sestupně podle podobnosti. Výsledkem vyhledávání je předem určený počet prvků ze začátku takto vzniklého seznamu. Princip můžeme znázornit digramem na obrázku 2.1.



Obrázek 2.1: Princip vyhledání podobných modelů v databázi, kde je dotazovaný model porovnán se všemi modely v databázi.



Obrázek 2.2: Princip vyhledávání s využitím deskriptorů. Diagram je rozdělen dle typu operací na *online* a *offline* část.

Stanovení podobnosti dvou 3D modelů, neboli jejich sesouhlasení, není triviální. Při zkoumání této problematiky vzniklo několik různých metod, které mají společný základní princip. Snaží se zjednodušit popis tvaru modelu do podoby například reálného čísla, vektorů čísel, histogramu, případně nějaké složitější datové struktury. Metody se tedy liší podobou a způsobem získání tohoto popisu modelu neboli deskriptoru.

Podobnost se tak nestanovuje z dat modelu, ale používají se k tomu právě deskriptory. Výhodou je, že deskriptor pro každý model stačí vypočítat pouze jednou. Diagram 2.2 znázorňuje operace prováděné při vyhledávání za použití deskriptorů. Dělíme je na:

- **online operace** - provádějí se při každém požadavku na vyhledávání a
- **offline operace** - provádějí se typicky pouze jednou, například při přidávání nového modelu do databáze.

2.2 Podobnost

Pro stanovení podobnosti s (anglicky *similarity*) dvou objektů je nutné vypočítat tzv. vzdálenost d (anglicky *destination*) mezi jejich deskriptory. Formálně vzdálenost d dvou objektů z množiny M definuje autor článku [7] jako funkci:

$$d\colon M \times M \to \mathbb{R}^+ \cup \{0\}. \tag{2.1}$$

Funkce by měla splňovat tyto požadavky:

- **identita:** pro všechna $x \in M, d(x,x) = 0$,
- **pozitivita:** pro všechna $x, y \in M, x \neq y, d(x,y) > 0$,
- **symetrie:** pro všechna $x, y \in M, d(x,y) = d(y,x)$,
- **trojúhelníková nerovnost:** pro všechna $x, y, z \in M, d(x,z) \leq d(x,y) + d(y,z)$,

- **invariantnost vůči transformacím:** nechť T je množina transformací, pak pro všechna $x, y \in S, t \in T, d(t(x), t(y)) = d(x, y)$.

Malá vzdálenost mezi deskriptory dvou objektů znamená velkou podobnost. Známe-li vzdálenost, počítá se podobnost podle vzorce:

$$s = \frac{1}{1+d}. \tag{2.2}$$

Z definice vzdálenosti (obor hodnot $\mathbb{R}^+ \cup \{0\}$) můžeme odvodit, že podobnost nabývá hodnot z intervalu $(0, 1\rangle$.

2.2.1 Euklidovská vzdálenost

Euklidovská vzdálenost d_E definuje vzdálenost bodů A a B v n-rozměrném prostoru vztahem:

$$d_E = |AB| = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}. \tag{2.3}$$

Vhodná je tedy pouze pro deskriptory ve tvaru bodu (vektoru) v n-rozměrném prostoru.

2.2.2 Kosinová podobnost

Kosinová podobnost s_K je míra podobnosti dvou nenulových n-rozměrných vektorů. Počítá se jako kosinus úhlu θ mezi těmito vektory pomocí vzorce:

$$s_K = cos(\theta) = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2}\sqrt{\sum_{i=1}^{n} {B_i}^2}} \tag{2.4}$$

2.3 Deskriptory pro popis 3D tvaru

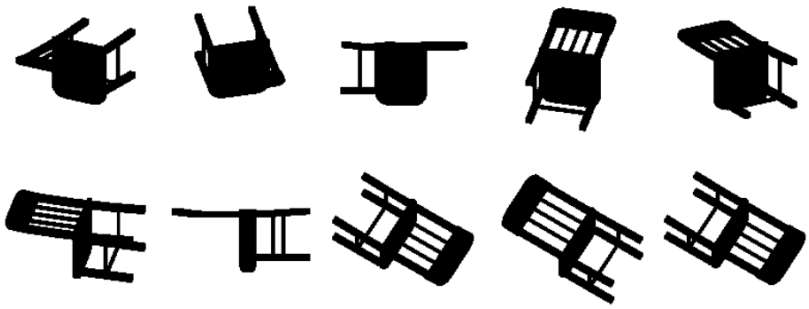
Již z názvu je patrné, že deskriptor by měl něco popisovat. V tomto případě se popis týká tvaru modelu. Ideálně by se měly deskriptory velmi rozdílných modelů lišit podstatně více, než deskriptory podobných modelů.

Dalším důležitým požadavkem je, aby byly invariantní vůči: rotacím, posunům a změně velikosti modelů. Pokud to neřeší přímo algoritmus pro výpočet deskriptoru, je nutné modely před výpočtem normalizovat.

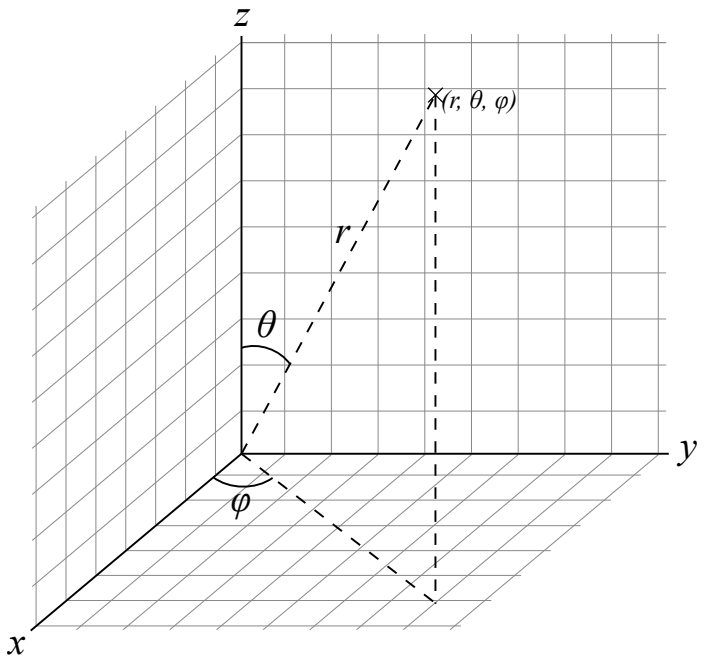
2.3.1 Lightfield deskriptor

Tento typ popisu je podobný způsobu znázornění 3D tvaru na papír. Takovéto ztvárnění trojrozměrného světa je pro člověka intuitivní. Využívá se zde projekce 3D tvaru do 2D roviny. Na rozdíl od promítání využívaného při vizualizaci modelů, je výstupní (zpravidla bitmapový) obraz prahován. Není tedy nutné využití stínovacích metod, textur, osvětlení a podobně. Zpravidla jde o černobílý obrázek, kde černou barvou je model a bílou pozadí, viz obrázek 2.3. Z toho vychází název „*Lightfield*“ tedy „světelná pole“.

Takto je model promítnut z různých úhlů, čímž vznikne sada světelných polí. Na porovnání těchto obrázků se pak používají metody pro počítačové vidění. Nevýhodou této metody je poměrně velká paměťová náročnost.



Obrázek 2.3: Příklad 3D tvaru promítnutého do roviny u metody *Lightfield deskriptorů* [1].



Obrázek 2.4: Ukázka bodu a jeho souřadnic r, θ a φ ve sférické soustavě souřadnic (zdroj http://wikipedia.org).

2.3.2 Spherical harmonics deskriptor

Tento deskriptor je založen na tzv. sférických harmonických funkcích. Více informací je možné získat z knihy [4]. Jsou to funkce definované na povrchu koule neboli ve sférické soustavě souřadnic.

Tato soustava je definovaná souřadnicí r, která udává vzdálenost bodu od počátku souřadnic. Druhá souřadnice se označuje θ a udává odklon průvodiče bodu od osy z. Třetí souřadnice φ pak udává odklon průvodiče od osy x. Bod v této soustavě a jeho souřadnice ukazuje obrázek 2.4.

Sférické harmonické funkce jsou definovány jako:

$$Y_l^m(\theta,\varphi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(cos\theta) e^{im\varphi}, \tag{2.5}$$

kde $P_l^m(z)$ jsou tzv. asociované Lendreho polynomy.

Metoda, jak pomocí sférických harmonických (anglicky *spherical harmonics*) extrahovat deskriptor 3D tvaru, je popsána v článku [3].

2.3.3 Zernikeho deskriptor

Výpočet *Zernikeho deskriptorů* je založen na tzv. Zernikeho polynomech, které jsou pojmenovány po nizozemském optikovi a fyzikovi Frederikovi Zernike. Ten je využil ve své práci [8] v oblasti optiky.

Jde o množinu ortogonálních polynomů definovaných na jednotkovém kruhu. Rozlišujeme liché a sudé Zernikeho polynomy. Sudé polynomy definuje rovnice:

$$Z_n^m(\rho, \varphi) = R_n^m(\rho) cos(m\rho) \tag{2.6}$$

a liché:

$$Z_n^{-m}(\rho, \varphi) = R_n^m(\rho) sin(m\rho), \tag{2.7}$$

kde m a n jsou nezáporná celá čísla, pro která platí $n \geq m$, φ je azimut a ρ je vzdálenost od středu kruhu. Jde o jednotkový kruh, z čehož vyplývá $0 \leq \rho \leq 1$. R_n^m je radiální polynom:

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k!\left(\frac{n+m}{2}-k\right)!\left(\frac{n-m}{2}-k\right)!} \rho^{n-2k}. \tag{2.8}$$

Metoda pro výpočet *3D Zernikeho deskriptorů* popsaná v článku [5] je rozšířením *Spherical harmonics* deskriptoru.

2.4 Dělení metod pro vyhledávání

Metody pro porovnávání 3D tvaru využívající deskriptory dělí článek [7] na tři kategorie. Metody založené na:

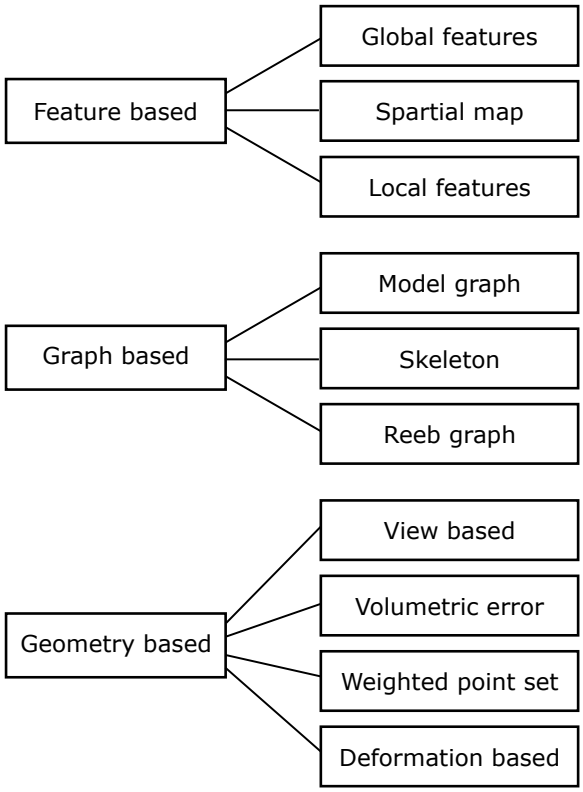
- vlastnostech (feature based),
- grafu (graph based) a
- geometrii (geometry based).

Kategorie je možné dále dělit, jak ukazuje obrázek 2.5. Protože u některých kategorií nelze název výstižně a stručně přeložit, bude se následující text držet anglických názvů. Nutno také poznamenat, že tyto kategorie nejsou disjunktní, a proto se některé metody řadí do více z nich.

2.4.1 Metody založené na vlastnostech

V angličtině se setkáme s výstižnějším názvem *feature based methods*. Deskriptory těchto metod mají tvar n-dimenzionálního vektoru, kde n bývá v řádu desítek až nízkých stovek dimenzí. Na deskriptor můžeme nahlížet jako na bod v mnohodimenzionálním prostoru a vzdálenost těchto bodů odpovídá podobnosti. Podobné modely se tedy nachází v okolí tohoto bodu.

Jeden model může mít pouze jeden deskriptor, což je případ metod z kategorií *globálních vlastností* (*global features*) a *spartial map*. A nebo více v případě *lokálních vlastností* (*local features*). Metody založené na *Spherical harmonics* a *3D Zernikeho deskriptorech* se řadí do kategorie tzv. *spartial map*.



Obrázek 2.5: Dělení metod porovnávání 3D tvaru založené na deskriptorech [7].

2.4.2 Metody založené na grafech

První kategorie bere v potaz celkovou geometrii modelu, kdežto „metody založené na grafech“ (*graph based methods*) model dekomponují na menší části a popisují, jak spolu souvisejí.

2.4.3 Metody založené na geometrii

Do této kategorie řadíme například *metody založené na pohledu* (*view based methods*). Ty jsou založeny na myšlence, že dva 3D modely vypadají podobně, pokud jsou podobné ze všech úhlů pohledu. Spadá sem například metoda založená na *Lightfield* deskriptorech, která je popsána v podkapitole 2.3.1.

Kapitola 3

Metriky kvality vyhledávání

Při tvorbě enginu pro vyhledávání a zejména při jeho optimalizaci jsou velmi podstatné tzv. metriky. Ty říkají, jak dobře engine vyhledává. Různé metriky se liší způsobem vyjádření této kvality. Může jít o graf, histogram, nebo reálné číslo. Teoretické poznatky z této kapitoly jsou využity pro vyhodnocení kvality navrženého a implementovaného enginu v kapitole číslo 6.

3.1 Vyhledávání

Pro pochopení metrik kvality vyhledávání musíme nejprve tento pojem.

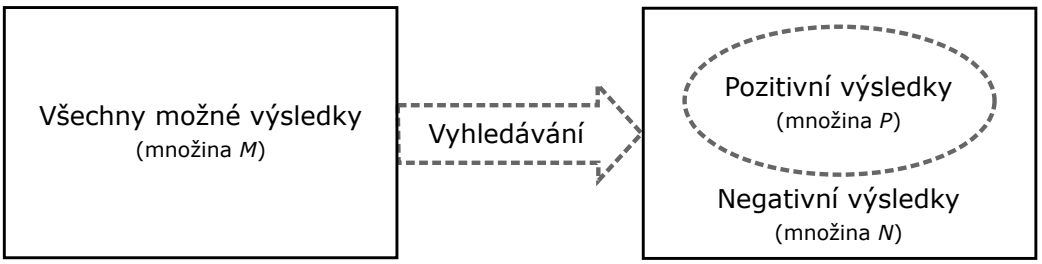
Vyhledávání je binární klasifikace, která dělí množinu všech vyhledatelných objektů (v našem případě množinu modelů) M do dvou podmnožin. Do množiny tzv. *pozitivních* výsledku P a množiny *negativních* výsledků N. Platí, že $M = P \cup N$, což ukazuje i obrázek 3.1.

V souvislosti s vyhledáváním se často uvádí pojem *práh* t (anglicky *threshold*). Ten se využívá v případě, kdy je vyhledávání založené na následujícím principu. Existuje tzv. *ohodnocovací funkce* $s\colon M \to \mathbb{R}$, která ohodnotí všechny prvky množiny M reálným číslem. Množiny P a N jsou pak definovány jako:

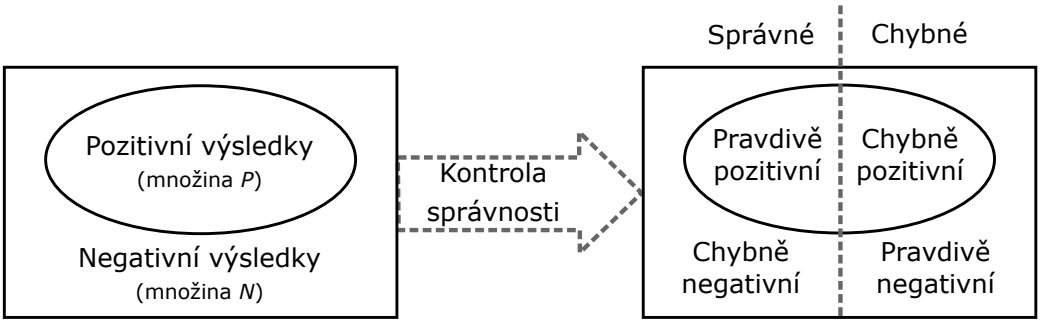
$$P = \{x \in M | s(x) > t\}, \tag{3.1}$$

$$N = \{x \in M | s(x) \leq t\}. \tag{3.2}$$

Předpokládáme, že vyhledávání není dokonalé. V tomto případě to znamená, že všechny objekty označené jako *pozitivní* nemusí nutně být *pravdivě pozitivní* (anglicky *true positive*). Těm nesprávně vyhledaným říkáme *falešně pozitivní* (anglicky *false positive*).



Obrázek 3.1: Ukázka vyhledávání, tedy klasifikace množiny M na podmnožiny P a N.



Obrázek 3.2: Znázornění případu, kdy známe správné výsledky vyhledávání a lze se zabývat kvalitou vyhledávání.

Stejně tak množina *negativních* výsledků se dělí na *pravdivě negativní* (anglicky *true negative*) a *falešně negativní* (anglicky *false negative*). Množiny znázorňuje obrázek 3.2.

Všechny metriky vyžadují, aby byly známy *správné* výsledky vyhledávání. Určit jaké výsledky jsou *správné* často nelze, proto bychom měli takové výsledky nazývat spíše *ideálními*. Tyto výsledky se určují empiricky.

3.1.1 Precision

$$precision = \frac{pravdivě\ pozitivní}{pozitivní} \tag{3.3}$$

Českým ekvivalentem slova *precision* je přesnost. Ze vzorce 3.3 lze vyčíst závislost mezi počtem správně nalezených položek a celkovým počtem nalezených položek. Z toho plyne, že *precision* bude rovno jedné v případě, že mezi *pozitivními* položkami budou pouze *pravdivě pozitivní*.

3.1.2 Recall

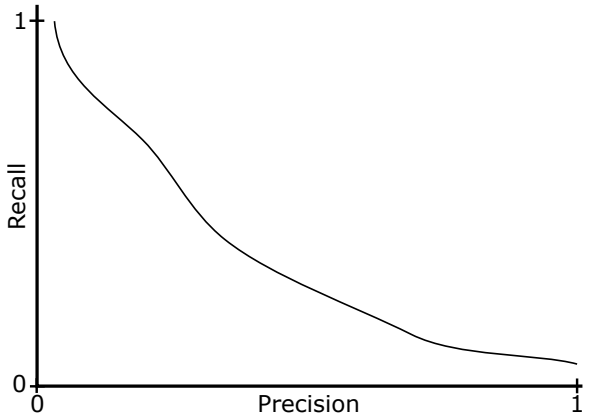
$$recall = \frac{pravdivě\ pozitivní}{(pravdivě\ pozitivní) + (falešně\ negativní)} \tag{3.4}$$

Do češtiny můžeme přeložit slovo *recall* jako citlivost. Je určen poměrem mezi počtem správně nalezených položek a celkovým počtem nalezitelných položek. Tedy *recall* je roven jedné v případě, že je množina *falešně negativní* prázdná.

3.2 Precision-recall křivka

Křivka znázorňuje závislost veličiny *precision* na *recall*. V praxi vypadá podobně jako na obrázku 3.3.

Jak vypočítat *precision* a *recall* víme, ale tím dostaneme pouze jeden bod této křivky. Vyjádření celé křivky docílíme postupnou změnou *prahové* hodnoty.



Obrázek 3.3: Příklad precision-recall křivky.

3.3 Average precision

Tato metrika vychází z *precison-recall* křivky. Jde vlastně o velikost plochy pod touto křivkou. Formálně ji popisuje vzorec:

$$ap = \int_0^1 p(r)dr, \tag{3.5}$$

kde p je funkce udávající *precision* na základě hodnoty *recall*.

3.4 Mean average precision

Mean average precision (*map*) vyjadřuje průměrnou hodnotu metriky *average precision* (*ap*) pro Q dotazů. Počítá se pomocí vzorce:

$$map = \frac{\sum_{q=1}^{Q} ap(q)}{Q}. \tag{3.6}$$

Kapitola 4

Návrh řešení vyhledávání v databázi 3D modelů

S využitím teorie popsané v předchozím textu, je v této kapitole vytvořen návrh řešení vyhledávání podobných 3D modelů v jednoduché databázi. Při návrhu je kladen důraz na logické členění programu a znovupoužitelnost. Aby se logické celky daly použít samostatně (například formou knihovny). Případně by měla být umožněna rozšiřitelnost funkcionalit bez nutnosti zasahovat do ostatních logických celků.

4.1 Požadavky

Nejprve je třeba si stanovit požadavky, kterých se bude návrh držet. Ty budou postupně rozvíjeny a konkretizovány.

- **Vyhledávání podobných modelů** lze považovat za klíčový požadavek výsledné aplikace, ale je potřeba jej více zpřesnit. Databáze bude obsahovat množinu objektů, kde každý z nich bude jeden 3D model. S tímto modelem se bude zacházet jako s celkem, takže i v případě, že jde o komplexní model (například model domu), bude brán v potaz celkový tvar tohoto modelu. Vyhledávání se nebude zabývat tím, že by se v tomto komplexním modelu hledaly pouze jeho části (například okno). Vyhledávácí engine je tedy navržen pro samostatné komponenty větších scén (automobil, člověk, rostlina, židle, ...). Zaměřuje se na využití při navrhování či modelování scén, kdy má uživatel k dispozici velkou databázi již vytvořených modelů, ze kterých čerpá. Funkcionalita vyhledání podobných modelů mu zjednoduší její procházení a výběr.
- **Vkládání, úprava a mazání modelů** jsou pro databázi nezbytné operace. Ke každému modelu budou uloženy doplňující informace, jako například *název*, *krátký popis*, nebo třeba *datum vytvoření*. Dle konkrétního zaměření by pak byly vhodné i další atributy popisující model.
- **Architektura klient-server** je u databází velmi častým řešením. Proto i navrhované řešení bude založena na stejném principu. Databázové servery nabízí velké množství funkcionalit, které zpřístupňuje například prostřednictvím jazyka SQL. Tento návrh uvažuje jednoduchou databázi s neměnnou strukturou, která bude umožňovat pouze základní operace. Komunikace bude probíhat přes navržené rozhraní, jímž se zabývá podkapitola 4.4. To by mělo být podobě jako jazyk SQL nezávislé na implementaci

serveru i klienta. Kromě toho bude možné využít také implementačně závislá rozhraní, protože návrh dbá i na znovupoužitelnost kódu. Části implementace pak bude možné použít jako knihovnu.

- **Náhled modelu formou obrázku** je dobrý způsob, jak člověku předat co nejvíce informací o modelu. Ještě interaktivnější je 3D náhled modelu, což je spíš požadavek na uživatelské rozhraní. K jeho zajištění na straně serveru stačí zpřístupnit data modelu ve vhodném formátu. Oba tyto požadavky jsou v návrhu zahrnuty.

- **Perzistence** je vlastnost, bez které databáze téměř postrádá smysl. S touto vlastností se úzce pojí pojem serializace. Serializovat je třeba modely s jejich atributy, a to do vhodného formátu.

- **Podpora více vstupních i výstupních formátů modelu** není problém, protože existuje mnoho knihoven a nástrojů na jejich konverzi. Navíc existují různé formáty a nelze stanovit ten nejčastěji používaný, proto se bude návrh zabývat podporou více formátů. Přidávání do databáze by mělo být nezávislé na formátu. Uživateli by mělo být umožněno nahrát model v libovolném formátu. Konkrétně návrh počítá s podporou formátů: *stl*, *obj* a *ply*.

  Pro zjednodušení ukládání a konverze modelů v databázi nebudou brány v potaz textury a další atributy, které některé formáty podporují. Návrh databáze bude počítat pouze s tvarem modelů.

Návrh se zabývá pouze okrajově klientem a jeho uživatelským rozhraním, proto konkrétní využití serveru není úzce vymezeno. Rozhraní je navrženo univerzálně, aby mohl být server použit například jako jádro webové služby, nebo jako plugin pro program užívaný k modelování scény. Každé z těchto využití může klást na návrh specifické požadavky a není možně je všechny splnit. Proto se návrh zabývá hlavně jejich společnými požadavky.

4.2 Struktura serverové části

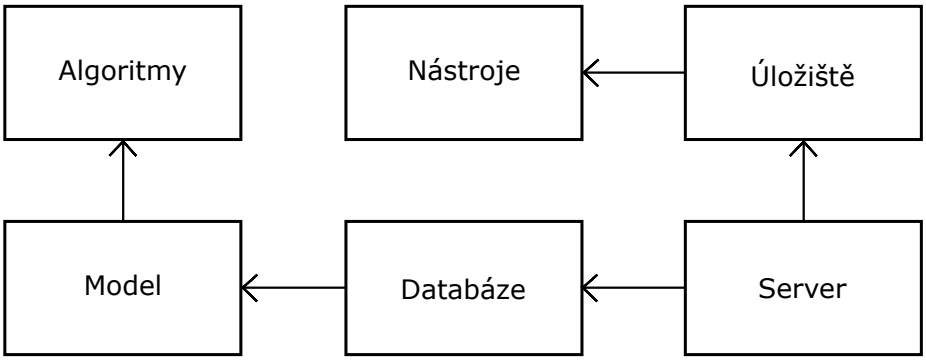
Návrh serveru můžeme rozdělit na menší, relativně samostatné části. Tyto moduly a jejich provázanost znázorňuje obrázek 4.1. Byla snaha o to, aby provázanost mezi moduly byla jednosměrná a nevznikaly kruhové závislosti. Díky splnění těchto podmínek lze například modul *model* používat spolu s modulem *algoritmy* samostatně.

Algoritmy

Modul má za úkol počítat algoritmy na výpočet podobnosti dvou deskriptorů. Minimálně by měl podporovat *kosinovou podobnost* a algoritmus pro určení *Euklidovské vzdálenosti*. Více informací o podobnosti a jejím výpočtu je v podkapitole 2.2.

Model

Zajišťuje abstrakci nad modelem. Hlavní úkol tohoto modulu je ukládat atributy modelu. K zajištění perzistence bude nutné umět tyto atributy serializovat a deserializovat. Vhodným formátem pro serializaci je například JSON. Díky tomu je možné metadata modelu uložit například do souboru při ukončování databáze, nebo je možné tuto serializaci využít při odesílání skrze komunikační rozhraní.



Obrázek 4.1: Diagram dekompozice do modulů a jejich vzájemná provázanost (šipka je ve směru od závislého modulu k jeho závislosti).

Abychom mohli vyhledávání založit na principu uvedeném v podkapitole 2.1, bude třeba určit podobnost dvou modelů. K tomu slouží závislý modul *algoritmy*.

Databáze

Tento modul zajišťuje hlavní funkce databáze. Obsahuje kolekci modelů se kterými umí manipulovat. Největší pozornost bude třeba věnovat výběru modelů z databáze. Zde bude nutné podporovat vyhledávání dle podobnosti. To zajišťuje vyhledávácí engine, jehož návrh více popisuje podkapitola 4.3.

Návrh odděluje *data* modelu od jeho *metadat*. Hlavní důvod je, že *metadata* bude výhodné držet v paměti, kdežto *data* by zabírala příliš mnoho místa a pro většinu operací nejsou potřeba. O *metadata* se stará právě tento modul.

Pro zajištění perzistence databáze se používá serializace všech modelů do souboru a jejich případná deserializace. Díky tomu je možné databázi uložit a později opět načíst ze souboru.

Nástroje

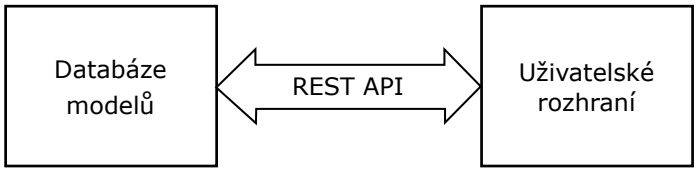
Zajišťuje korektní propojení nástrojů třetích stran, jako jsou knihovny a nástroje, se zbytkem aplikace. Jde například o knihovny pro výpočet deskriptorů, konverzi a tvorbu obrázkových náhledů modelu. Dále modul zajišťuje podporu různých platforem z důvodu různého umístění knihoven a různého způsobu využití.

K dispozici je nástroj pro výpočet *3D Zernikeho deskriptorů* a *Spherical harmonics* deskriptorů. Konkrétní nástroje a knihovny udává podkapitola 5.2.

Úložiště

Tento modul zajišťuje přístup k datům modelů. Ty bude třeba číst, ukládat, konvertovat, počítat z nich deskriptory a vytvářet náhledy modelu. Poslední tři zmíněné funkce zprostředkovává modul *nástroje*, proto je zde patrná závislost.

Data konkrétního modelu jsou určena jednoznačným identifikátorem a požadovaným formátem. Podporované jsou dle požadavků formáty: *stl*, *obj* a *ply*. Pro načítání dat jsou rovněž podporovány formáty *png* a *jpg*, kde jde o náhled modelu.



Obrázek 4.2: Příklad použití REST aplikačního rozhraní propojující databázi modelů s uživatelským rozhraním.

Server

Server zajišťuje komunikaci pomocí navrženého rozhraní. Požadavky pak předává buď *databázi* nebo *úložišti* v závislosti na typu požadavku. Návrh toho implementačně nezávislého rozhraní je popsán v podkapitole 4.4.

4.3 Návrh vyhledávání

Stěžejní funkcí navrhovaného serveru je vyhledávání 3D modelů podle podobnosti. Tento vyhledávácí engine je součástí navržené databáze. Toto uzké spojení je výhodné použitému principu, který více popisuje podkapitola 2.1.

Princip lze ve zkratce shrnout tak, že pro každý model je extrahován zvolenou metodou deskriptor, který bude uložen v databázi. Extrakce deskriptoru je tzv. *offline* operace, takže se provádí pouze jednou, konkrétně při přidání modelu do databáze. Při vyhledávání je pak nutné porovnat deskriptor dotazovaného modelu s deskriptory všech modelů v databázi. Iterace nad všemi modely je častou operací. Oddělení databáze od vyhledávácího enginu by způsobovalo zbytečné zpomalení operace.

Navrhované řešení se zabývá metodou *Spherical harmonics* a metodou tzv. *3D Zernikeho deskriptorů*. K extrakci deskriptorů pomocí těchto metod jsou použity knihovny třetích stran, které přesněji specifikuje podkapitola 5.2.

4.4 Rozhraní serveru

Jedním z požadavků na návrh je implementačně nezávislé rozhraní. Abychom jeho návrh odstínili od implementačních detailů, jako je programovací jazyk nebo použitá platforma, je nutné rozhraní definovat na úrovni síťové komunikace. Konktrétně půjde o aplikační vrstvu ISO/OSI modelu.

Pro potřeby databáze, která odpovídá na jednoduché požadavky klienta (klient vždy iniciuje komunikaci), je vhodný aplikační protokol HTTP. Jako serializační formát byl zvolen JSON pro svoji rozšířenost a podporu ve všech běžně používaných programovacích jazycích.

Protože záznamy databáze bude možné vkládat, číst, upravovat i mazat, je vhodné použít k tomuto účelu navržený přístup. Tím je například REST, který byl definován v práci [2, Kapitola 5] a jde o způsob využití protokolu HTTP. Obrázek 4.2 ukazuje využití databáze 3D modelů skrze navržené rozhraní. Je zde patrné, že uživatelské rozhraní je nezávislé na implementaci databáze.

Rozhraní současně zpřístupňuje funkcionality vyhledávácího enginu, který je součástí navržené databáze.

Při návrhu REST aplikačního rozhraní (API) musí být určeno, jaké tzv. zdroje bude zprostředkovávat. Protože jde o databázi modelů, bude jeden zdroj nazvaný `models`. Z pohledu rozhraní je vhodné oddělit samotná data modelu od metadat; zdroj `models` reprezentuje pouze informace o modelu. Samotná data reprezentuje zdroj `files`, který slouží pro stahování dat a náhledu jednotlivých modelů.

Dalším navrhovaným zdrojem je `similars`, jehož jedinou funkcionalitou je návrat podobných modelů. Zde si naopak vystačíme s metodou POST, pomocí níž se pošlou data modelu, pro který se vrátí odpověď obsahující seznam podobných modelů. V tomto případě nedojde k uložení modelu do databáze.

Podrobná dokumentace rozhraní je součástí příloh (příloha A). Následující text pouze shrnuje jeho klíčové vlastnosti.

Zdroj `models`

Tento zdroj zpřístupňuje metadata modelu, používá k tomu HTTP metody:

- GET pro čtení,
- POST pro vkládání,
- PUT pro úpravu a
- DELETE pro mazání.

Při získávání zdroje skrze REST API, je ve zvyku umožnit získání kolekce několika položek, nebo právě jedné položky. Pokud vybíráme jednu položku, musíme ji nějak identifikovat, což děláme zpravidla celočíselným identifikátorem `id`. V případě kolekce se udávají omezující parametry jako *maximální počet* (`limit`) nebo *id první položky* (`start`). Kromě toho umožňuje výběr kolekce modelů podobných k požadovanému modelu. Argument se nazývá `similar` a jako hodnota je očekáváno `id` daného modelu. V tomto případě vrací rozhraní modely rozšířené o atribut `similarity`, udávající podobnost.

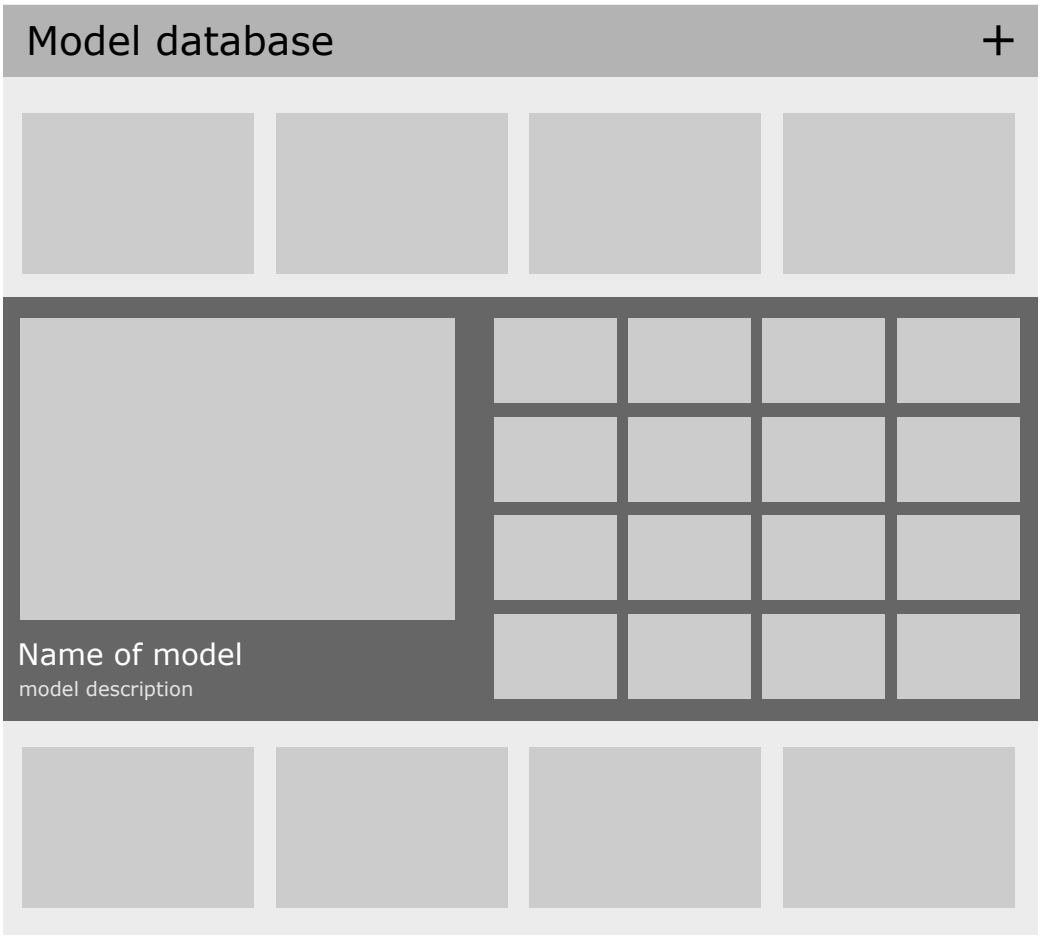
U požadavků na jeden konkrétní model se osvědčilo přidat atribut `similars`, který obsahuje kolekci podobných modelů. Tím není třeba dělat zvlášť požadavek na model a na podobné modely.

Požadavek typu POST slouží k vytvoření nového záznamu v databázi. Povinnými argumenty jsou `name` pro název a `file` pro data modelu. Požadavek PUT je určen k aktualizaci konkrétního záznamu, definovaného pomocí `id`. Tento požadavek již nemá povinné argumenty. Argument `file` není podporován, tedy data modelu nelze měnit.

Zdroj `files`

Zde si vystačíme s metodou GET. Zdroj slouží k získávání dat modelů a ke stahování jejich náhledů. O jaký model se jedná určuje název souboru.

Dochází zde k poměrně velkým datovým přenosům, a proto by v případě většího počtu uživatelů bylo vhodné pro tuto funkcionalitu využít nějaké vhodnější řešení. Příkladem je třeba Apache HTTP server. Rozhraní je k tomuto problému uzpůsobeno tím, že zdroj `models` má u každého modelu atribut `preview`, který obsahuje URL obrázku s náhledem modelu a atribut `data`, jenž obsahuje kolekci URL adres odkazující na platná data k tomuto modelu (v různých formátech). Tedy je možné, aby tyto URL odkazovaly i mimo API. Klidně na různé servery kvůli rozložení zátěže.



Obrázek 4.3: Návrh rozložení webového rozhraní ukázkového klienta pro databázi 3D modelů.

Zdroj `similars`

Ten rozšiřuje funkcionalitu zdroje `models`. Podporuje pouze metodu POST a má povinný argument `file`. Slouží k odeslání dat modelu podobně jako POST u zdroje `models`, s tím rozdílem, že model není uložen do databáze. Požadavek vrátí odpověď ve stejném formátu jako GET u detailu modelu. Odpověď obsahuje atribut `similars` a kolekci podobných modelů. Tato funkcionalita se hodí k rychlému vyhledávání podobných modelů pro požadovaný soubor. Byla navržena pro rychlou demonstraci funkcionality databáze.

4.5 Návrh ukázkového klienta

Klient musí implementovat REST rozhraní navržené v podkapitole 4.4. Podrobnější informace o rozhraní jsou uvedeny v příloze A.

Cílem klienta je jednoduchým způsobem demonstrovat hlavní funkce databáze a zejména vyhledávácího enginu. Z toho důvodu byly vybrány webové technologie, které jsou přenositelné prakticky na jakoukoliv platformu a nevyžadují žádné instalování a zprovozňování. Konkrétně jde o HTML5, CSS3 a JavaScript.

Výchozí stav aplikace by měl zobrazovat výčet několika modelů z databáze, který se bude dle potřeby rozšiřovat. Pro ilustraci, o jaký model se jedná, stačí zobrazovat malé náhledy. Po kliknuti na jakýkoliv obrázek se otevře karta s detailem modelu, kde bude kromě většího obrázku také *název modelu*, *popis modelu* a další dostupné informace. V podobné mřížce, v jaké se nachází celkový výčet modelů, se zobrazí menší výčet s podobnými modely.

Inspirací při návrhu rozhraní byly vyhledávače obrázků (například Google obrázky), které při náhledu konkrétního obrázku zobrazí i výčet jemu podobných obrázků. Navržené rozložení je naznačeno na obrázku 4.3.

Pro vyhledání podobných modelů se zde nabízí využití funkce „drag and drop“. Po přetáhnutí souboru s modelem se automaticky vyhledají jemu podobné modely. Uživatelské rozhraní na to zareaguje tím, že otevře detail tohoto modelu.

Dále bude možné vkládání nových modelů do databáze. To se vyvolá tlačítkem „+“ v horní liště. Tím se otevře formulář pro vyplnění *názvu modelu* a *popisu modelu*. Soubor s *daty* modelu se přidá opět pomocí „drag and drop“.

Kapitola 5

Implementace

Kapitola popisuje implementační detaily řešení navrženého v předchozí kapitole.

Ačkoliv jsou server a klient na sobě nezávislí a spojuje je pouze komunikace přes aplikační rozhraní (REST API), je součástí serveru databáze modelů i jednoduchý HTTP server. Ten zprostředkovává statické soubory webového klienta (*html*, *css* a *js* soubory). Důvod tohoto spojení je především to, že webové prohlížeče z bezpečnostních důvodu nedovolují XHR (XML Http Request) u lokálně otevřeného *html* souboru. Navíc mají omezení i na požadavky na jinou doménu, než ze které *html* soubor pochází. Toto lze obejít povolením pomocí HTTP hlavičky.

5.1 Technologie

Nejprve bude třeba rozhodnout, jaké technologie budou využity pro implementaci. Zadání výběr zužuje na C/C++ a Python. Rozhodl jsem se pro využít obou technologií současně hlavně proto, že každá z nich má svoje výhody, ale i nevýhody. Pro výpočty deskriptorů a obecně práci s daty modelů se využívají programy napsané v C++. Pro implementaci databáze, vyhledávacího enginu a REST API serveru je pak využit Python, výhodný svojí dynamičností a širokou škálou předpřipravených knihoven. Použit je Python 3. Vývoj probíhal konkrétně na verzi 3.6.0.

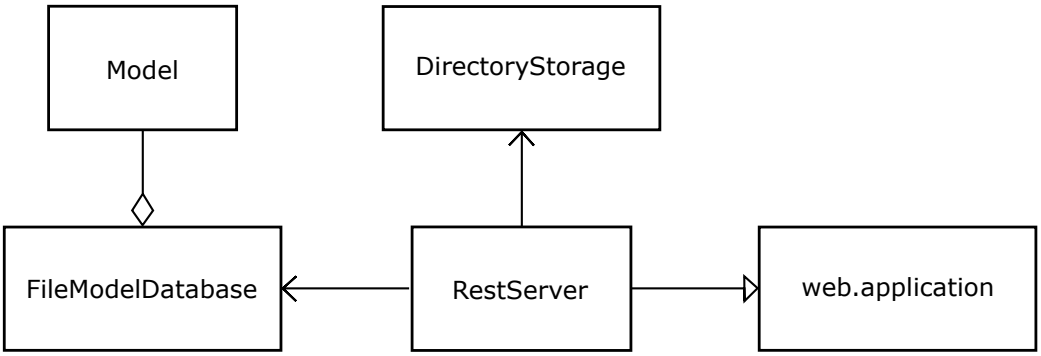
Pro podporu znovupoužitelnosti kódu se využívají balíčky Pythonu (anglicky *packages*). Přesněji kód databáze modelů je součástí jednoho balíčku `model_db`.

5.2 Knihovny a nástroje třetích stran

Při implementaci hrají velkou roli použité knihovny a nástroje. Existuje několik implementovaných knihoven pro extrakci deskriptorů. Tato práce využívá knihovnu na výpočet *3D Zernikeho deskriptorů*[1] popsanou v článku [6]. Případně je možné využít knihovnu pro výpočet *Spherical harmonics*[2] deskriptorů, pospanou v článku [3]. U obou knihoven bylo upraveno rozhraní pro snadnější volání z Pythonu.

Dále je využit nástroj *meshconv*[3] pro konverzi modelů. Tento nástroj byl zvolen pro jeho jednouché rozhraní a osvědčil se svojí bezproblémovostí.

[1] https://github.com/Sunwinds/ShapeDescriptor
[2] https://github.com/mkazhdan/ShapeSPH
[3] http://www.patrickmin.com/meshconv/



Obrázek 5.1: Základní diagram tříd databáze 3D modelů s REST rozhraním.

Pro generování náhledů k 3D modelům se využívá program *Blender*[4]. Ten umožnuje používat i pokročilé funkce skrze předaný Python skript. Generování probíhá na pozadí a není potřeba žádná interakce s UI.

Pro usnadnění tvorby HTTP serveru je využita knihovna *web.py*[5]. Ta se využívá jak pro REST API, taky pro zpřístupnění webového klienta.

Obě metody pro výpočet deskriptoru vyžadují voxelizovaný model. K tomuto účelu se používá nastroj *binvox*[6].

5.3 Atributy modelu

Každý model, uložený v databázi, je definován atributy:

- `id` - jednoznačný celočíselný identifikátor modelu,
- `name` - název modelu,
- `description` - textový popis modelu,
- `added_date` - časové razítko přidání do databáze,
- `descriptor` - vektor deskriptoru,
- `file_id` - id pro získání dat modelu (odpovídá názvu souboru).

5.4 Důležité třídy databáze

Python mimo jiné podporuje objektově orientované paradigma (OOP), které je v dnešní době v oblibě a i zde bude využito. Tato podkapitola popisuje přehled základních tříd, jaké navržená databáze využívá. Jejich vzájemné vztahy ukazuje diagram tříd 5.1. Třídy vycházejí z modulů navržených v kapitole 4.2 a dále je konkretizují.

[4] https://www.blender.org/
[5] http://webpy.org/
[6] http://www.patrickmin.com/binvox/

`Model`

Třída reprezentuje jeden model uložený v databázi. Atributy modelu jsou definované v podkapitole 5.3. K určení podobnosti dvou modelů je metoda `how_similar`, která vrací podobnost z intervalu od 0 do 1. Podobnost je počítána z atributu `descriptor` buď pomocí *kosinové podobnosti* nebo pomocí *Euklidovské vzdálenosti*.

`FileModelDatabase`

Tato třída obsahuje seznam modelů a umožňuje práci s nimi, respektive s jejich metadaty. Stará se o jejich přidávání, mazání a čtení na základě různých požadavků. Metody pro výběr z databáze umožnují následující:

- výběr **jednoho modelu** na základě *id*,
- výběr **kolekce modelů** začínající modelem s konkrétním *id* a s maximálním *počtem* modelů,
- výběr **kolekce modelů** na základě podobnosti k modelu s určitým *id*.

Pro splnění požadavku perzistence definuje třída metody pro uložení a načtení ze souboru. Formát pro uložení byl zvolen opět JSON.

`DirectoryStorage`

Třída `FileModelDatabase` se stará o ukládání metadat k modelům. Za samotná data je zodpovědná třída `DirectoryStorage`. Každý model je uložen v souboru ve formátu *stl*. Tato třída na základě `file_id`:

- ukládá data modelu,
- čte data modelu,
- získává deskriptor modelu.

Každá instance má specifikovanou složku pro *modely* (`model_dir`) a také složku pro *dočasné soubory* (`temp_dir`). Jako dočasné soubory se považují například náhledy modelů nebo modely v jiném formátu než *stl*, tedy soubory, které jsme schopni opakovaně vygenerovat z uloženého modelu. Funguje to vlastně jako cache.

Mimo to třída umožňuje vypočítat deskriptor pro konkrétní `file_id` vybranou metodou. Samotný výpočet neřeší tato třída, ale funkce definované v modulu `algorithms`.

`RestServer`

Tato třída vytváří HTTP server, na kterém zpřístupňuje REST rozhraní definované v podkapitole 4.4. Třída dědí od třídy `web.application` z knihovny *web.py*. Její funkčnost rozšiřuje především o obsluhu *url* adres, které odpovídají REST API.

Kapitola 6

Dosažené výsledky

Následující kapitola shrnuje výsledky dosažené při implementaci databáze, kterou popisuje kapitola 5. Informuje o způsobu vyhodnocení těchto výsledků. Současně jsou zde uvedeny možnosti dalšího vývoje založené na poznatcích získaných při návrhu a implementaci současného řešení.

6.1 Použité datové sady

Pro otestování kvality vyhledávání je potřeba tzv. datová sada (můžeme se setkat i s pojmem benchmark). V našem případě obsahující 3D modely. Tyto modely musíme buď ručně klasifikovat, nebo použít již klasifikovanou sadu. Klasifikací je myšleno roztřídění modelů do tříd, kde každá obsahuje jen podobné modely. Příkladem může být třída *automobily*, *lidé* a podobně.

Použity byly hned dvě datové sady. Tzv. *Princeton Shape Benchmark*[1] (*PSB*) a *NTU 3D Model Benchmark*[2] verze 1. Obě zmíněné datové sady jsou klasifikované a tabulka 6.1 ukazuje jejich bližší specifikaci. Na obrázku 6.1 jsou ukázky několika modelů z těchto datových sad a ukázky tříd do kterých jsou klasifikovány.

	PSB	NTU
Počet klasifikovaných modelů	907	549
Počet tříd	131	107

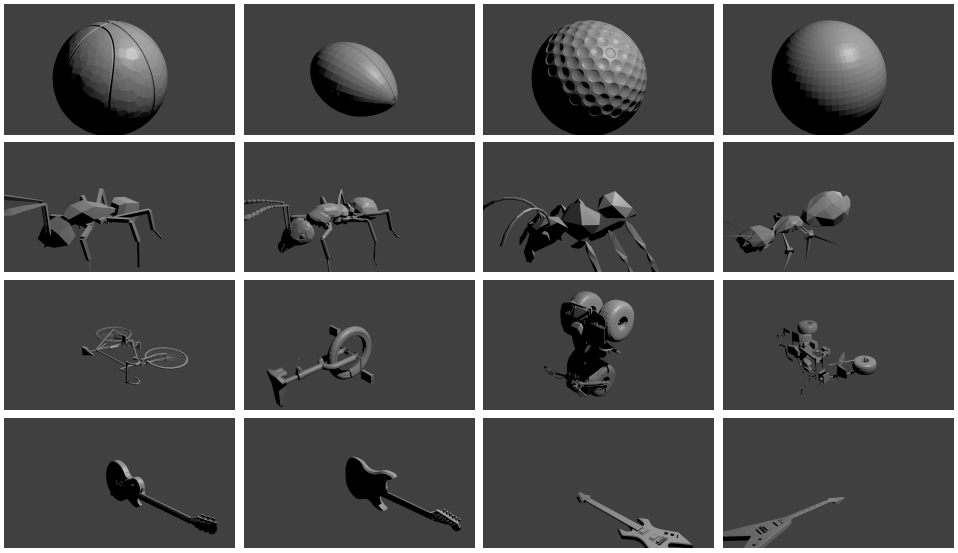
Tabulka 6.1: Přehled využitých datových sad.

6.2 Způsob získávání metrik

Aby se daly snadno vyhodnotit metriky popsané v kapitole 3, byl vytvořen nástroj navržený přímo pro implementovanou databázi. Jde o skript napsaný v Pythonu. Pro ukázku znovupoužití kódu využívá nástroj balíček `model_db`. Jeho dokumentace je v příloze B.

Skript vyžaduje databázi naplněnou klasifikovanými modely. To, do jaké třídy model patří, nástroj určuje podle popisu modelu (atribut `description`). Takže dva modely patří do stejné třídy, pokud mají stejný popis (například „automobil“).

[1] http://shape.cs.princeton.edu/benchmark/
[2] http://3d.csie.ntu.edu.tw/~dynamic/database/



Obrázek 6.1: Ukázka modelů z datových sad a jejich klasifikace (první řada: třída *ball*, druhá řada: třída *ant*, třetí řada: třída *bike*, čtvrtá řada: třída *electrical guitar*).

Nástroj metriky vyhodnocuje tím způsobem, že pro každý model v databázi vyhodnotí podobnost k ostatním modelům. Poté je pomocí změny prahové hodnoty (jak popisuje podkapitola 3.1) stanoven průběh precision-recall křivky. Z ní lze vyvodit další metriky.

Algoritmus má kvadratickou asymptotickou časovou složitost. Rychlost provádění také závisí na počtu dimenzí deskriptorů.

6.3 Experimenty

V tuto chvíli máme nástroj, který nám dokáže ohodnotit kvalitu vyhledávání. Je možné s implementací experimentovat a kontrolovat, zda se kvalita zlepšuje či zhoršuje.

Při experimentech se ukázalo jako nejvýhodnější využít metriku *mean average precision* (*map*). Hlavním důvodem je, že bere v potaz více dotazů současně, viz definice v podkapitole 3.4.

6.3.1 Srovnání algoritmů výpočtu podobnosti

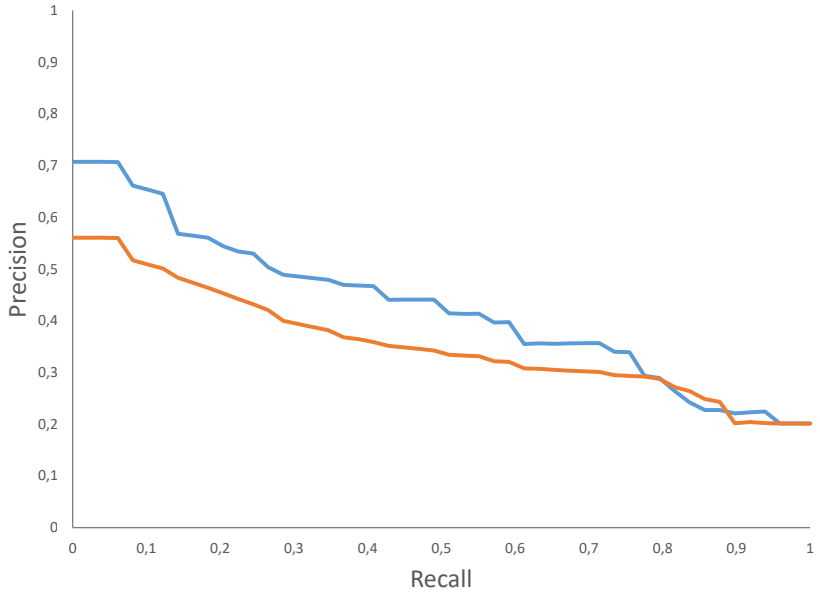
Experiment porovnává dva algoritmy pro výpočet podobnosti uvedené v podkapitole 2.2, konkrétně *kosinovou podobnost* a algoritmus využívající *Euklidovské vzdálenosti*.

Výsledkem experimentu jsou *precision-recall* křivky na obrázku 6.2, které ukazují, že *kosinová podobnost* vykazuje lepší výsledky. Díky tomuto zjištění je algoritmus *kosinové podobnosti* používán jako výchozí.

6.3.2 Optimalizace výpočtu Zernikeho deskriptorů

Tento experiment si klade za cíl nastavit výpočet *3D Zernikeho deskriptorů* tak, aby při vyhledávání podával engin přesnější výsledky. Při celém experimentu se k provnávání deskriptorů využívá *kosinová podobnost*.

Vstupem nástroje pro výpočet *Zernikeho deskriptoru* je voxelizovaný model. Díky tomu je nutné optimalizovat proměnnou r, která reprezentuje rozlišení voxelizace.



Obrázek 6.2: Precision-recall křivky, křivka výše: při využití *kosinové podobnosti* a pod ní: křivka *Euklidovské vzdálenosti*.

Knihovna pro *Zernikeho deskriptory* umožňuje nastavení maximálního řádu Zernikeho momentů. To bude druhá proměnná pro optimalizaci a značit ji budeme N.

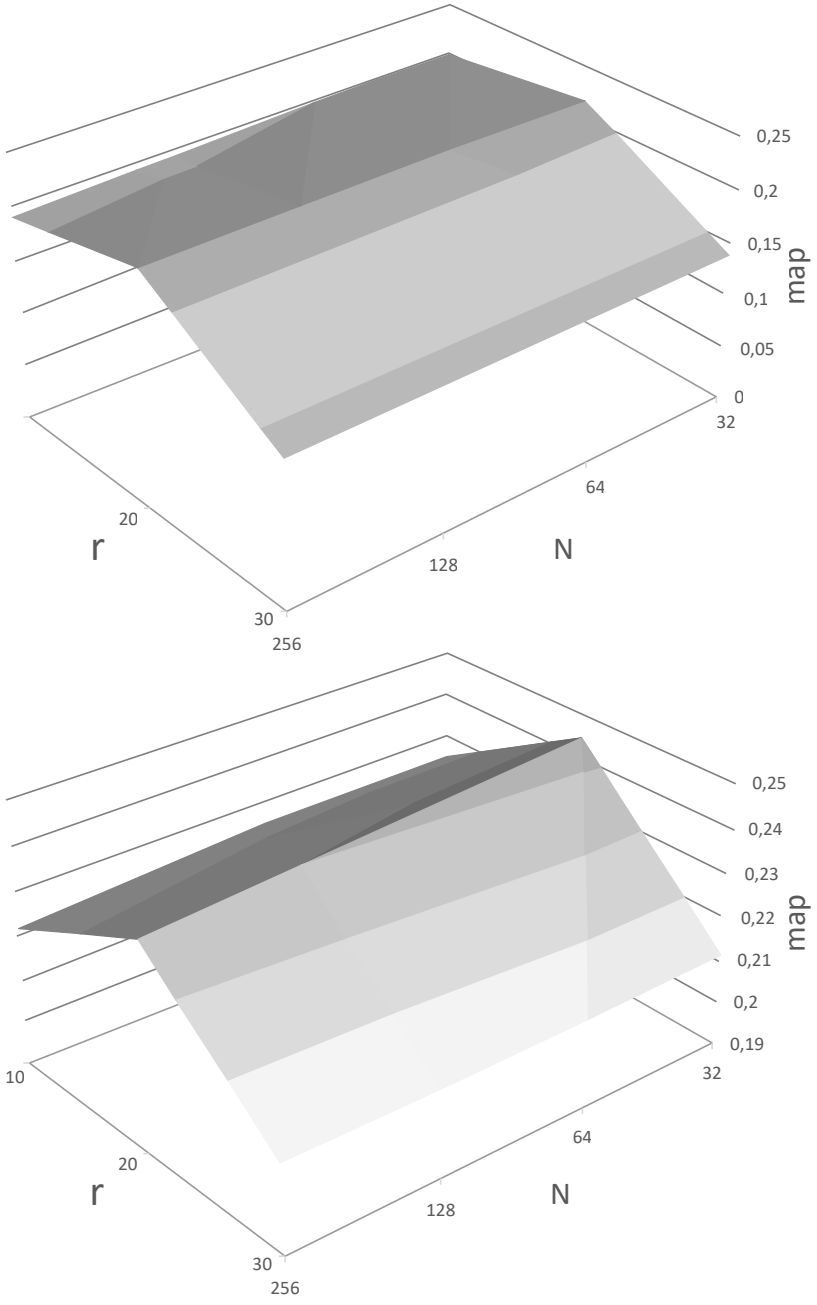
Jako počáteční hodnoty proměnných byly zvoleny 256 pro r a 20 pro N. Jedná se totiž o výchozí hodnoty použitých nástrojů.

Pro různé kombinace r a N byla vypočítána metrika *map* a výsledné hodnoty zobrazují grafy 6.3. První graf vznikl použitím PSB datové sady, druhý použitím NTU benchmarku. Na grafu nás zajímá extrém, kde zvolená metrika vyhodnotila největší kvalitu vyhledávání na zvolené datové sadě. Za optimální hodnoty proměnných můžeme považovat $r = 32$ a $N = 20$. Současně je vidět drobný rozdíl mezi použitými datovými sadami, proto nemůžeme brát tyto optimalizované hodnoty jako univerzální pro všechny modely.

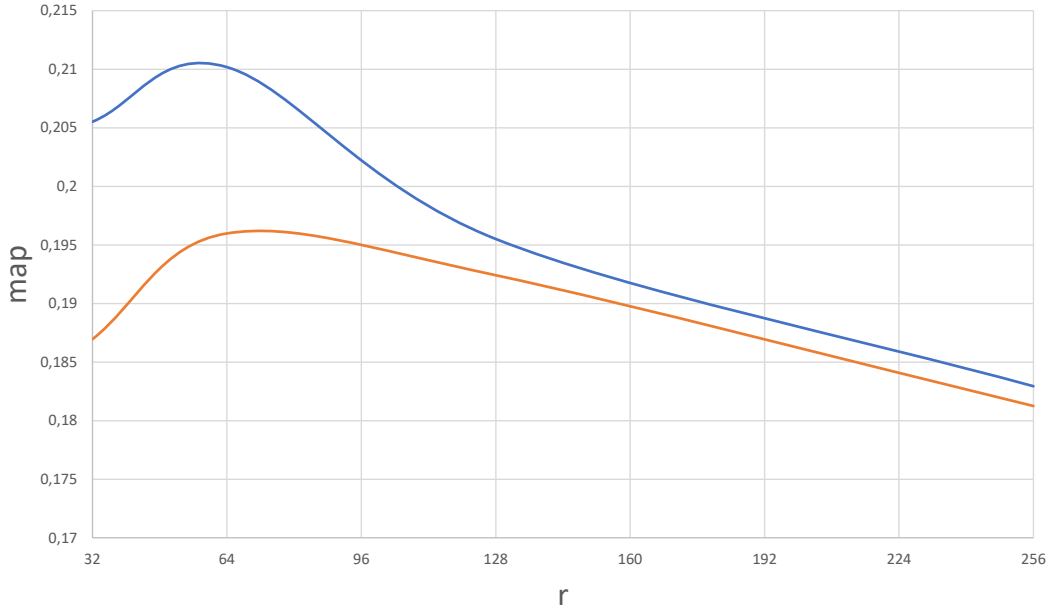
6.3.3 Optimalizace výpočtu Spherical harmonics

Tento experiment je velmi podobný tomu předchozímu s tím rozdílem, že zde je zkoumána pouze závislost rozlišení voxelizace r na *mean average precision* (*map*). Výslednou závislost ukazuje graf 6.4.

Na grafu je patrné maximum poblíž hodnoty $r = 64$, tedy blízký výsledek jako u předchozího experimentu. Tato podobnost lze pravděpodobně vysvětlit tím, že metoda *Zernikeho deskriptorů* vychází z *Spherical harmonics*.



Obrázek 6.3: Grafy závislosti metriky *map* při použití metody *3D Zernikeho deskriptorů* na rozlišení voxelizace r a maximálním řádu Zernikeho polynomů N. Horní graf vznikl použitím PSB datové sady, druhý použitím NTU benchmarku.



Obrázek 6.4: Grafy závislosti metriky *map* při použití metody *Spherical harmonics* na rozlišení voxelizace r. Horní křivka patří k datové sadě *NTU* a spodní k *PSB*.

6.4 Možnosti dalšího vývoje

Výsledná databáze 3D modelů implementuje pouze základní operace a je zde prostor na rozšíření o nové funkcionality. Například návrh neřeší otázku autentizace a autorizace uživatelů.

Dále z hlediska implementace by bylo dobré lépe propojit databázi psanou v Pythonu s nástroji pro výpočet deskriptorů psanými v C++. V současnosti se používá modul, který umožňuje spouštět nové procesy (`subprocess`) a číst jejich výstup. Přitom Python umožňuje přímo vytvářet moduly v C++. Díky této možnosti by bylo vše součástí jednoho balíčku, který by bylo možné snadněji distribuovat. Výhodou by byla také snadnější přenositelnost na jiné platformy.

Asi nejdůležitější otázkou je, jakým způsobem lze vylepšovat engine pro vyhledávání podobných modelů. V této oblasti lze experimentovat hlavně s výpočtem deskriptorů, ať už vylepšováním stávajících metod, nebo vymýšlením nových.

Jinou variantou zlepšení výsledků vyhledávání je ponechat současně implementované metody a experimentovat s předzpracováním 3D modelů. Tím je myšlena například dekompozice modelů na menší části, kdy pro každou část bude vypočítán deskriptor. Vyhledávání pak bude brát v potaz podobnost těchto různých částí, jejich počet a vzájemnou polohu. Tato metoda by vyžadovala mnohem sofistikovanější algoritmy pro porovnávání těchto N-tic deskriptorů. Dekompozice modelů není rovněž triviální záležitost.

Kapitola 7

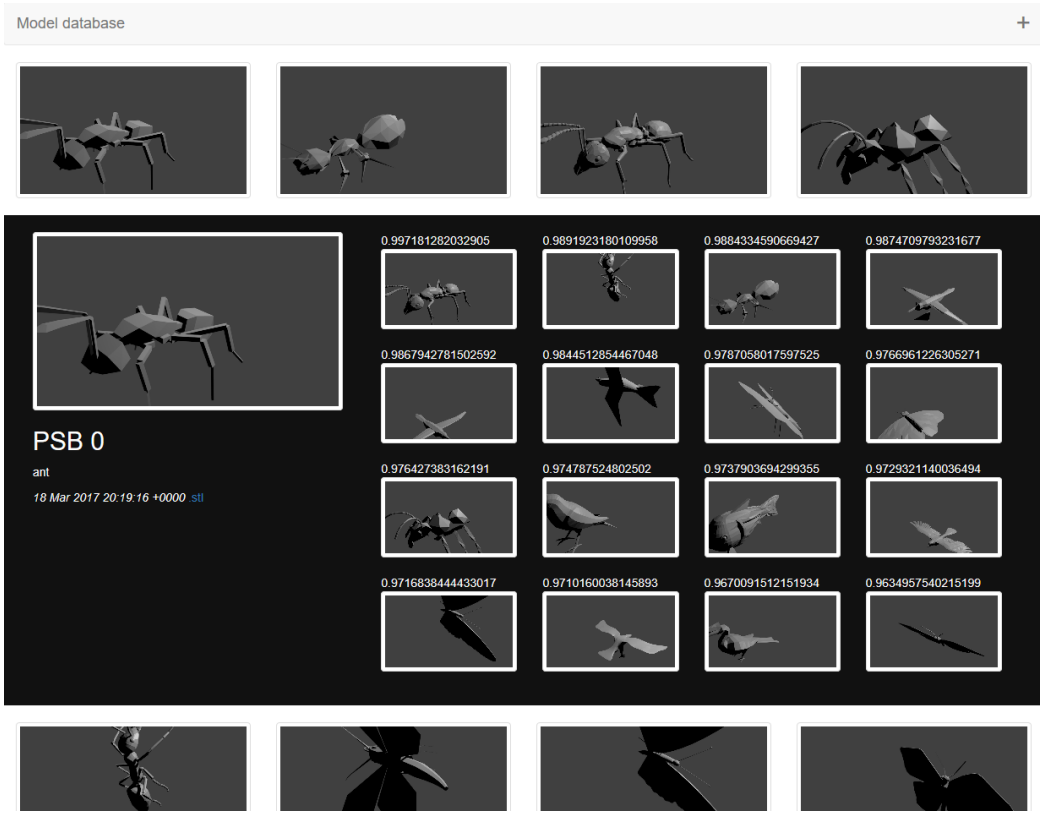
Závěr

Výsledkem práce je v Pythonu implementovaný engine pro vyhledávání podobných 3D modelů v jednoduché databázi. Databáze s enginem je v roli serveru a komunikuje prostřednictvím navrženého REST API. Pro demonstraci funkčnosti vyhledávání podobných 3D modelů byl implementován jednoduchý webový klient.

Výhodou řešení je univerzální rozhraní REST API využívající serializační formát JSON. Tato kombinace je velmi dobře podporovaná v různých programovacích jazycích a klienta je možné implementovat pro nejrůznější platformy.

Návrh a implementace vyžadovaly nastudování a pochopení metod využívaných pro porovnání a určení podobnosti 3D modelů. A také metod pro extrakci a porovnání deskriptorů.

Výsledné řešení lze využít k různým účelům, kde je potřeba uložit velké množství 3D modelů. Jejich procházení je zjednodušeno díky funkci vyhledání podobných modelů. Na obrázku 7.1 je zachyceno webové rozhraní, které demonstruje engin vyhledávající podobné 3D modely.



Obrázek 7.1: Snímek obrazovky demonstračního webového rozhraní se zobrazeným detailem modelu a náhledy podobných modelů.

Literatura

[1] Chen, D.-Y.; Tian, X.-P.; Shen, Y.-T.; aj.: On Visual Similarity Based 3D Model Retrieval. *Computer Graphics Forum*, ročník 22, č. 3, 2003: s. 223–232, ISSN 1467-8659, doi:10.1111/1467-8659.00669.
URL http://dx.doi.org/10.1111/1467-8659.00669

[2] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, University of California, 2000.
URL http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[3] Kazhdan, M.; Funkhouser, T.; Rusinkiewicz, S.: Rotation Invariant Spherical Harmonic Representation of 3D Shape Descriptors. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '03, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, ISBN 1-58113-687-0, s. 156–164.
URL http://dl.acm.org/citation.cfm?id=882370.882392

[4] MacRobert, T.: *Spherical Harmonics: An Elementary Treatise on Harmonic Functions, with Applications*. Chronic illness in the United States, Dover Publ., 1948.
URL https://books.google.cz/books?id=n_pQAAAAMAAJ

[5] Novotni, M.; Klein, R.: Shape retrieval using 3D Zernike descriptors. *Computer-Aided Design*, ročník 36, č. 11, 2004: s. 1047 – 1062, ISSN 0010-4485, solid Modeling Theory and Applications.
URL http://www.sciencedirect.com/science/article/pii/S0010448504000077

[6] Novotni, M.; Klein, R.: Shape retrieval using 3D Zernike descriptors. *Computer-Aided Design*, ročník 36, č. 11, 2004: s. 1047 – 1062, ISSN 0010-4485, doi:http://doi.org/10.1016/j.cad.2004.01.005, solid Modeling Theory and Applications.
URL http://www.sciencedirect.com/science/article/pii/S0010448504000077

[7] Tangelder, J. W. H.; Veltkamp, R. C.: A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications*, ročník 39, č. 3, 2007: str. 441, ISSN 1573-7721, doi:10.1007/s11042-007-0181-0.
URL http://dx.doi.org/10.1007/s11042-007-0181-0

[8] Zernike, v. F.: Beugungstheorie des schneidenver-fahrens und seiner verbesserten form, der phasenkontrastmethode. *Physica*, ročník 1, Květen 1934: s. 689–704, doi:10.1016/S0031-8914(34)80259-5.

Přílohy

Příloha A

Dokumentace REST API

Součástí práce je návrh implementačně nezávislého rozhraní typu REST. Jako serializační formát je použit JSON. Všechny URL rozhraní mají prefix `/api`. Ten lze dle potřeby změnit.

Argumenty vyznačené **tučně** jsou povinné, ostatní pouze volitelné. Opakující se části těla odpovědi jsou zestručněny pomocí „...“. Některé navržené funkcionality nejsou implementovány a jsou zde uvedeny pouze pro úplnost.

GET `/api/models`

Argumenty

`start`	`id` počátečního modelu (výchozí 1)
`limit`	maximální počet vrácených modelů (výchozí 20)
`similar`	hodnota je `id` modelu, vůči kterému budou hledány podobné modely (pokud je použit, tak se argument `start` ignoruje)
`query`	hledaný řetězec pro fulltextové vyhledávání (*neimplementováno*)

Odpověď

```
[
  {
    "id": 1,
    "name": "Rabbit",
    "description": "Model of rabbit",
    "added_date": "18 Mar 2017 20:19:16 +0000",
    "preview": "/files/d99fefd6-8f83-4097-988e-d183fd04292d.png",
    "data": [
      "/files/d99fefd6-8f83-4097-988e-d183fd04292d.stl",
      ...
    ],
    "similarity": 0.9831877343175236
  },
  ...
]
```

Atribut `similarity` je v odpovědi pouze při použití požadavku s argumentem `similar`.

GET `/api/models/{id}`

Argumenty

`id`	jedinečný celočíselný identifikátor modelu
`similars`	počet požadovaných podobných modelů (výchozí 0)

Odpověď

```
{
  "id": 1,
  "name": "Rabbit",
  "description": "Model of rabbit",
  "added_date": "18 Mar 2017 20:19:23 +0000",
  "preview": "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.png",
  "data": [
    "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.stl",
    ...
  ]
  "similars": [
    {
      "id": 18,
      ...
      "similarity": 0.9831877343175236
    },
    ...
  ]
}
```

V případě hodnoty 0 u argumentu `similars` chybí v odpovědi atribut `similars`.

POST `/api/models`

Argumenty

`name`	název modelu
`file`	data modelu
`description`	popis
`format`	vstupní formát modelu (výchozí *.stl*)

Odpověď

```
{
  "id": 1,
  "name": "Rabbit",
  "description": "Model of rabbit",
  "added_date": "18 Mar 2017 20:19:23 +0000",
  "preview": "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.png",
  "data": [
    "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.stl",
    ...
  ]
}
```

PUT `/api/models/{id}`

Argumenty

`id`	jedinečný celočíselný identifikátor modelu
`name`	název modelu
`description`	popis

Odpověď

```
{
  "id": 1,
  "name": "Rabbit",
  "description": "Model of rabbit",
  "added_date": "18 Mar 2017 20:19:23 +0000",
  "preview": "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.png",
  "data": [
    "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.stl",
    ...
  ]
}
```

DELETE `/api/models/{id}`

Argumenty

`id`	jedinečný celočíselný identifikátor modelu

Odpověď

Odpověď nemá tělo, úspěšnost operace se rozlišuje návratovým kódem HTTP.

POST `/api/similars`

Slouží k vyhledání podobných modelů bez nutnosti model vkládat do databáze.

Argumenty

`file` data modelu
`format` vstupní formát modelu (výchozí *.stl*)

Odpověď

```
{
  "id": 0,
  "name": "",
  "description": "",
  "added_date": "24 May 2017 20:35:23 +0000",
  "preview": "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.png",
  "data": [
    "/files/1fef9641-21b5-4e02-aac0-2addefdc3280.stl",
    ...
  ]
  "similars": [
    {
      "id": 25,
      ...
      "similarity": 0.943157734
    },
    ...
  ]
}
```

GET `/api/files/{file_id}.{format}`

Argumenty

`file_id` jedinečný identifikátor souboru
`format` formát souboru

Odpověď

Vrácena jsou data modelu v požadovaném formátu, pokud je podporován a `file_id` je správné.

Příloha B

Návody k použití výsledných programů

Tato příloha je dokumentací k implementované databázi a k dalším vytvořeným nástrojům. Všechny popisované programy se spouští pomocí příkazové řádky a mají celou řadu možných přepínačů a argumentů.

Databáze modelů

Jak již bylo v zmíněno v kapitole 5, databáze je implementovaná v jazyce Python. Vstupní bod databázového serveru je skript `model_db.py`. Tento skript používá balíček `model_db`, který obsahuje samotnou implementaci databáze a další potřebné moduly a třídy popsané v podkapitolách 4.2 a 5.4. Použití spouštěcího skriptu je:

```
./model_db.py database_file models_dir temp_dir [-m MODE] [-p PORT] [-o OUTPUT]
[-t THREADS] [-a ALGORITHM] [-s SIZE] [-n N] [-h] [-r] [-d] [-u] [-e]
```

`database_file`	cesta k JSON souboru s databází
`models_dir`	cesta ke složce pro ukládání modelů
`temp_dir`	cesta ke složce pro pro dočasné soubory
`-m -mode`	mód databáze (výchozí mód je `server`)
`-p -port`	pro změnu portu, na kterém server naslouchá (výchozí je 8080)
`-o -output`	nastaví cestu k souboru, kam bude uložena databáze (výchozí je `database_file`)
`-t -threads`	nastaví počet vláken pro výpočet deskriptorů (výchozí jsou 4)
`-a -algorithm`	nastaví algoritmus pro výpočet deskriptorů (výchozí je `zernike`, možno přepnout na `spherical`)
`-s -size`	rozlišení při voxelizaci (výchozí je 128)
`-N -N`	nastavuje maximální řád Zernikeho momentů při použití algoritmu `zernike`
`-h -help`	zobrazí nápovědu
`-r -readonly`	režim pouze pro čtení (nelze jakkoliv měnit databázi)
`-d -debug`	ladící režim (tiskne do konzole pomocné výpisy)
`-u -ui`	zpřístupnění uživatelského rozhraní prostřednictvím jednoduchého HTTP serveru

`-e -euclidean` pro porovnávání deskriptorů se použije Euklidovská podobnost (výchozí je kosinová)

Módy databáze

`server` databáze bude naslouchat na REST API
`check` zkontroluje deskriptory databáze a vypíše záznamy s chybějícím deskriptorem
`clear` vymaže u všech modelů deskriptory
`repair` vypočítá chybějící deskriptory
`recount` přepočítá všechny deskriptory

Výpočet metrik

Tento balíček (`retrieval_metrics`) v Pythonu slouží k vyhodnocení metrik určující kvalitu vyhledávání databáze. Skript `retrieval_metrics.py` tento modul využívá a je vstupním bodem pro vyhodnocování metrik. Kromě toho je pro správnou funkci skriptu a balíčku `retrieval_metrics` potřeba balíček `model_db`. Užití skriptu je:

```
./retrieval_metrics.py database_file metric_type [-e]
```

`database_file` cesta k JSON souboru s databází
`metric_type` typ požadované metriky
`-e -euclidean` pro porovnávání deskriptorů se použije Euklidovská podobnost (výchozí je kosinová)

Podporované typy metriky

`mc` tiskne pouze počet modelů v databázi
`pr` tiskne dvojice hodnot: *precision*, *recall*
`prp` tiskne dvojice hodnot: *precision*, *recall* a vykreslí je v podobě *precision-recall* křivky
`ap` tiskne dvojice hodnot: *id*, *average precision*
`map` tiskne metriku *mean average precision*, ve formátu: *počet modelů*, *mean average precision*, TOTAL
`amap` tiskne metriku *mean average precision* pro každou třídu zvlášť, ve formátu: *počet modelů*, *mean average precision*, *třída*
`all` vypočíta všechny podporované metriky

Výpočet deskriptorů

Pro výpočet deskriptorů je možné použít dva zde dokumentované nástroje: `ZernikeDescriptor` a `SphericalDescriptor`. Ačkoliv jde o díla třetích stran, bylo u obou upraveno vstupní a výstupní rozhraní pro jednotné použití a tato dokumentace popisuje jejich využití:

```
./ZernikeDescriptor binvox_file [max_order]
./SphericalDescriptor binvox_file
```

`binvox_file`	cesta k souboru s voxelizovaným modelem (formát binvox)
`max_order`	maximální řád Zernikeho polynomu (výchozí je 20)

Výstupní deskriptor je tisknut na standartní výstup na jeden řádek (oddělovačem je tabulátor) ve formátu:

```
binvox_file d0 d1 d2...dX
```